GENERALIZATION AND EFFICIENT IMPLEMENTATION OF CC4 NEURAL NETWORK

By

GANGASANI SUMANTH KUMAR REDDY

Master of Science in Computer Science

Oklahoma State University

Stillwater, Oklahoma

2008

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2008

GENERALIZATION AND EFFICIENT

IMPLEMENTATION OF CC4 NEURAL NETWORK

Thesis Approved:

Dr. Subhash Kak

Thesis Adviser

Dr. Nohpill Park

Dr. Venkatesh Sarangan

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

I take this opportunity to express my gratitude to everyone who is involved in the successful completion of my thesis.

It has been an honor to have Dr. Subhash Kak as my adviser and research supervisor. I am grateful to him for his advice, support and guidance, not just in the course of my research but also for my graduate career at OSU.

I would also like to thank Dr. Venkatesh Sarangan and Dr. Nohpill Park for being my thesis committee members and for providing valuable inputs to my research. I would also like to thank Abhishek Parakh, Pradeep Koduru, Lakshmi Prasad Banala and Sandeep Chalasani who helped me in refining my thesis. I thank Rajesh Kumar Magham, Arun Chavali, Yugendra Reddy Guvvala and Niranjan Koudi for their friendship and support. I thank everyone else who helped me in making this work possible.

Finally I am grateful eternallly to my family: my mother, father and my brother for their affection, care and support, at all times throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

CHAPTER I

INTRODUCTION

There has been tremendous improvement in computing technologies, but human beings are still considered to be more intelligent than any of the computers that exist. In fact, current computers cannot perform many tasks that human brain performs with ease. In other areas, however, even the desktop computer is a lot faster than the human brain. These are the areas where logic is involved. Performing arithmetic operations on huge numbers on a large scale is a fairly simple task for a digital computer that exists today. The same task is however very difficult and tedious a human. So, how do we claim that humans are more intelligent than computers? Consider some other tasks like solving a crossword puzzle. This task involves a lot of guesswork and intuition. Solutions to such puzzles come out of some idea or recollection based on the obscurely worded clue. Many human beings manage to get to the solutions to easy puzzles of this kind with the help of good clues. Computers, on the other hand, are very bad at solving such puzzles, even the easy ones.

Now, consider some basic tasks like vision and hearing which look perfectly logical to human beings. Human beings can look at things and figure out what they are almost instantly. They can even recognize things that are worn out or are out of shape without

any problem in most of the cases. Computers are not at all good at such tasks. After using some complex algorithms, huge resources and considerable amount of time, it is often seen that computers are not accurate enough at recognizing or categorizing things based on vision. Another simple and common task for humans is picking up objects and moving them to some place. Performing such tasks with computers requires use of exceptionally complex techniques. Even with such complex techniques the results are not always impressive. It has also been discussed, based on evidence from physical and biological sciences that machines do not self-organize as the brain does [9].

One obvious question that comes to everyone's mind is that though computers do many tasks with astonishing speeds and accuracy, why is it that they perform so badly in many other tasks that human beings or animals do? The answer to this question might lie in the nature of the design of computers. The architecture and organization of electronic components is not similar to that of the basic units in human brain.

Human Brain and Computer

When we look inside a computer, we see a lot of electronic components interconnected in some orderly fashion. We see chips and other basic components put on to a circuit board and interconnected by what are called as tracks. No such structural order is seen when looked inside a human or animal brain. The human brain looks like a grey homogenous mass on initial sight [13]. On thorough investigation, it is found to be one of the most complicated things man has ever encountered. The operation of human brain has not yet

been clear. However, researchers have identified that different functions are performed in different regions of the brain.

The brain works in a parallel fashion in contrary to the operation of most of the computers today. Although there are different kinds of parallel architectures for building computers, the degree of parallelism in human or animal brain exceeds the degree of parallelism in modern computer system architectures by leaps and bounds. Computers operate by executing a set of binary instruction in a serial fashion in most of the architectures. The processor in these systems can execute millions of such binary instructions in a second. The brain, in contrast, has a huge number of very basic or dumb processing units called nervous cells, also called neurons, which are highly interconnected with others.

The Biological Neuron:

The biological neuron is the basic unit or the core component of the brain. These units process and transmit information by means of chemical signals. The structure of these units is fairly simple to understand. The soma is the central part of the neuron and contains the nucleus of the cell. The size of the nucleus ranges from 3 to 18 micrometers in diameter. The input to the nucleus of the cell mostly happens through the cellular extensions with many branches called dendrites. The output of the neuron is carried through a cable-like projection that is finer than the dendrites. This projection can extend up to tens of thousands of times the diameter of the soma in its length. Most neurons have only one axon which, generally, is extensively branched and connected to the other target

cells to enable communication. Axon hillock is the part of the neuron where axon is attached to the soma. This has the highest density of voltage-dependent channels. The terminals of the axon are connected to the dendrites of the other target neurons at a special junction called the synapse. Axon terminal releases a chemical that is absorbed by the dendrite and converted in to an electrical signal which is given as input to the soma.

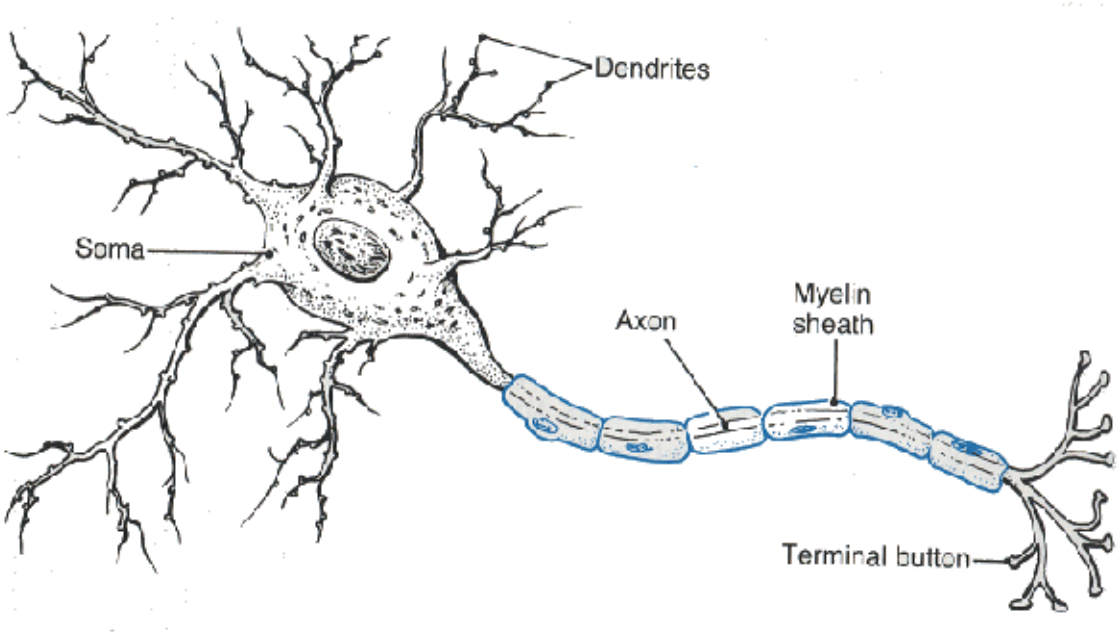


Figure 1.1: Biological Neuron (*Courtesy: Cedar Crest College*)

The Artificial Neuron:

An artificial neuron is an abstract mathematical model of a biological neuron. The first of its kind was the Threshold Logic Unit which was proposed by Warren McCulloch and Walter Pitts in 1943 [14]. This model of neuron can also receive one or more inputs (simulating dendrites) and generates an output (axon in biological neuron). These inputs are usually associated with weights (analogous to synaptic strengths in biological neurons) and the weighted sum of these inputs is used as input to the threshold function, also generally known as activation function or transfer function, to generate the output.

Later, different models for neurons were proposed which used different activation functions which include the signum and sigmoidal functions. The basic structure of an artificial neuron is given in figure 1.2.

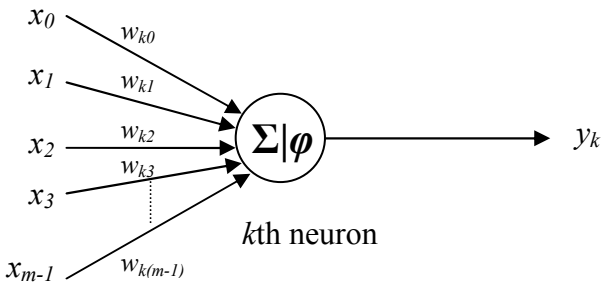


Figure 1.2: Artificial Neuron

Here, the output of the k^{th} neuron, y_k, is given by the following equation:

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

where, φ is the transition function, x is the input vector of size m and w_{kj} is the weight assigned to the link between the k^{th} neuron and the j^{th} input in the input vector.

CHAPTER II

INSTANTANEOUS NEURAL NETWORKS

INTRODUCTION

An artificial neural network can be defined as a network of unidirectional connections connecting very simple processing units, which may have a small amount of memory. These networks are generally motivated by the neuron interconnection network in the human brain. Such networks are being used in many areas like virtual reality, data compression, adaptive control, detection and tracking of moving targets etc. due to the improved efficiency and performance of these networks over conventional methods.

One such network is the CC4 network which is based on the corner classification approach to artificial neural network training which was proposed by Subhash Kak in 1992 [1] and was subsequently granted a U.S. patent. Although there are other techniques which include the back propagation algorithm for training a neural network, those are time-consuming and require substantial training.

CORNER CLASSIFICATION APPROACH

The basic idea behind this approach is to classify the outputs of the training samples to the corners of a multi-dimensional cube based on the corresponding inputs. Of the

four algorithms that exist in this class, CC4 is the most advanced.

Structure of CC4 Neural Network

The CC4 network is a three layered feed forward network of the basic binary neurons which use the threshold logic as the activation function. The three layers are:

i) Input layer

ii) Hidden layer

iii) Output layer

Input Layer

The CC4 network takes a *unary code* of the inputs as the input vector. Hence, every input is separately converted into its unary code before being fed to the network as input. Therefore, the number of input neurons required for representing these inputs in the input vector is the sum of the ranges of the inputs in the input vector.

In addition to these inputs we have a bias neuron that always takes the input as 1. Thus, the number of neurons in the input layer is one more than the sum of the ranges of the inputs in the input vector.

Hidden Layer

Every neuron in this layer corresponds to a single training sample in the training set. As a consequence, the number of binary neurons in the hidden layer is equal to the size of the training set. Each neuron in this layer is connected to all the neurons in the input layer, i.e. the neurons in the input layer and hidden layer are fully connected.

Output Layer

The neurons in this layer generate the output of the network. The number of neurons in this layer is equal to the minimum number of bits required to represent any output in the data set in the range of outputs in binary. Similar to the input layer and the hidden layer, the hidden layer and output layer are also fully connected. The general structure of the CC4 network is shown in Figure 2.1.

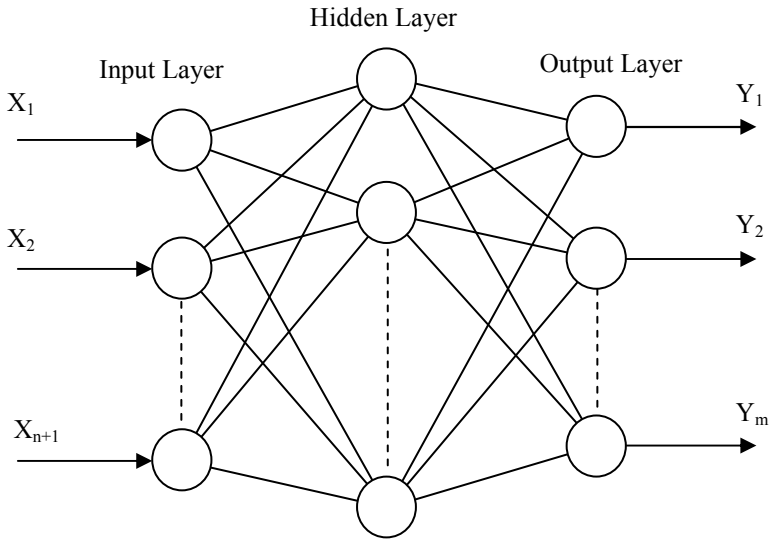


Figure 2.1: General structure of a CC4 neural network [1][11]

CC4 Functioning

The CC4 algorithm operates based on two new ideas which enable the network to learn and generalize. Learning is the process of assigning weights to the connections between the three layers of neurons. For this network, this is done by merely inspecting the input and output vectors of the training samples in the training set and assigning weights to the links between corresponding neurons. This learning process is called *prescriptive learning*. Another simple idea in this approach is the concept of *radius of generalization*.

This helps in classification of input vectors based on the class of stored vectors. If the hamming distance between the new input vector and any of the stored vectors is less than or equal to the user-specified radius, the outputs of all such stored vectors is considered for generating the output of the input vector. The number of 1s and 0s in every bit location of the output vector of all these stored vectors is calculated and added up. If the result is positive, the corresponding output neuron outputs 1 otherwise the output is 0.

The CC4 algorithm

The input and output weights to the hidden neurons are prescribed simply by inspecting the training samples. If an input neuron receives a 1, the weight of the link between the input neuron and respective hidden neuron is set to 1. If the input received is 0, the weight is set to -1. Similarly, based on the outputs in the output vector the weights between the hidden neuron and the corresponding output are set to 1 or -1. The extra input neuron, also called as bias neuron, is dealt with differently. The number of 1's in the input vector of the training sample are counted as *s* and the weight between the bias neuron and the hidden neuron corresponding to this input vector is set to *r*-*s*+1, where *r* is the user-specified radius of generalization. In short the weights for the links between input layer and the hidden neurons are assigned based on the following equation.

$$w_i[j] = \begin{cases} 1 & if \;\; x_i[j] = 1 \\ -1 & if \;\; x_i[j] = 0 \\ r - s + 1 & if \;\;\; j = n \end{cases}$$

The weights to the links between the hidden layer neurons and the output neurons is also set based on the same idea excepting that there is no extra bias neuron as in the input layer.

The formal CC4 algorithm for training of the neural network is as below [1][11]:

```
for each training vector x_i [n] do        //n = length of vector including bias
        s_i = no. of 1's in x_i[1:n-1];
        for j = 1 to n-1 do
                if x_i[j] = 1 then
                        w_i[j] = 1;        // Input weight to the hidden neuron
                else
                        w_i[j] = -1;
                end
        end
        w_i[n] = r-s_i+1;                  // r = radius of generalization
        for k=1 to m do
                if y_i[k] =1 then
                        u_i[k] = 1;        // Output weight to the hidden neuron
                else
                        u_i[k] = -1;
                end
        end
end
```

After training the neural network, the network is ready for generating outputs. New input vectors are fed to the input layer of the neural network. This vector along with the bias neuron, whose input is always 1, forms a single row matrix. The values of the prescribed weights are taken as another matrix. To get the inputs to the hidden neurons we multiply these two matrices and the product is the input to the hidden layer. All the positive values in the products are replaced with 1 and others are assigned with 0. This is due to the threshold function of the neurons, which is the step function whose threshold value is 0. Now a similar procedure is followed between the hidden layer and output layer to get the final output for the given input vector.

It has already been shown that this algorithm performs very well when compared to the back propagation algorithm in terms of speed [1][4]. However, we observe that the performance of the CC4 algorithm can be enhanced greatly in terms of speed. The

resources required can also be reduced and the radius of generalization can also be dynamic.

CC4 implementation issues

Since CC4 algorithm for training the neural networks uses *unary coding* for inputs, the number of input neurons depends directly on the range of the integers used in the input vector. Hence, higher the range more is the number of input neurons. Another problem with this algorithm is that if the training is memory consuming i.e. if the number of training samples is too many, the number of hidden neurons increases in a direct proportion. Both of these issues directly effect on the number of connections between the input layer and hidden layer. Also, due to the increase in the number of hidden neurons, the matrix for representing the weights between the hidden layer and output layer also increases in order. We already know that the time complexity of matrix multiplication is of cubic order. Hence, this makes the application of CC4 networks burdensome for a huge training set. Further, due to the increase in the number of input neurons with increase in range of inputs and hidden neurons with the intensity of training, this algorithm requires more memory to store the training samples which makes it difficult for this algorithm to be implemented on devices that are memory constrained. Static radius of generalization is another issue with insufficient training as the output is always 0 if there is no training sample within the radius of generalization.

CHAPTER III

A NEW AND EFFICIENT APPROACH USING NEURONS

The previous chapter lists several difficulties involved in efficiently implementing the CC4 algorithm with huge chunks of data. One option for efficient implementation is parallelizing the algorithm for use on multiple machines or a grid. This would substantially increase the throughput, but at the cost of increased resources. The complexity of this algorithm is of cubic order due to multiple matrix multiplications and increases with increase in the range of inputs in the input vector. The Fast Classification network, proposed by Kun Won Tang and Subhash Kak, has addressed the issues involved with the CC4 algorithm for real valued inputs [5]. The FPGA implementation of this Fast Classification network has also been discussed [10].

Generalized CC4 Neural Network

Here, we present a new instantaneous neural network that takes integers as inputs and gives the binary representation of the generated values as outputs. The new network also implements the two basic ideas of the CC4 networks, i.e. prescriptive learning and radius of generalization. In addition, this network also implements the concept of dynamic radius of generalization which can be defined by the user. The general structure of a basic network is given in figure 3.1.

This network, similar to the CC4 network, has three layers of neurons, namely, input neurons, hidden neurons and output neurons. Input neurons are the neurons that take integers as inputs. The number of input neurons depends on the problem specification, i.e. the number of integers required to represent the input vector of each sample in the data set. The output neurons produce outputs in binary representation of integer values. There is one extra neuron in the output layer that gives feedback to all the hidden neurons based on the results from the hidden layer and problem specification. The input neurons and output neurons are fully connected by a set of hidden neurons. The number of hidden neurons is equal to the number of training samples used for training the network.

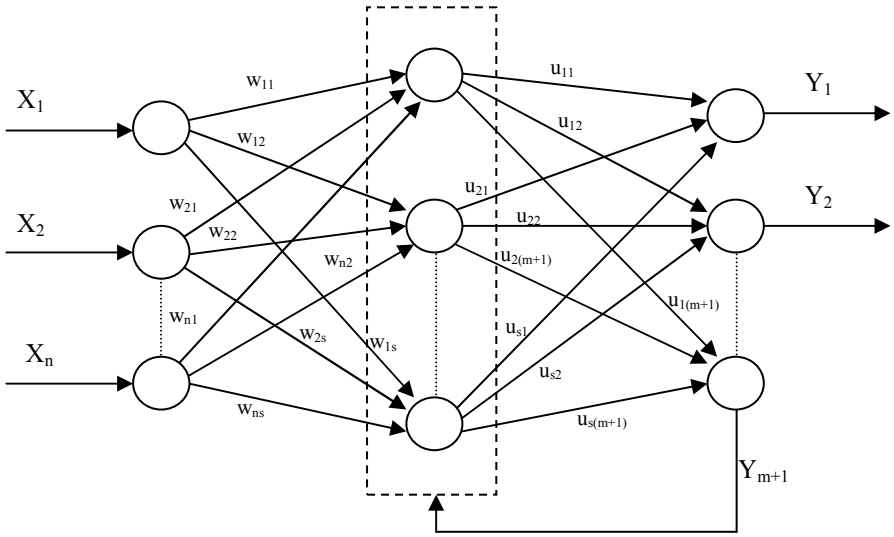


Figure 3.1: Basic structure of Generalized CC4 Neural Network

The weight associated to the links between hidden neurons and the feedback neuron in the output layer is always 1. This means all the neurons give some input to the hidden neuron at all times. Depending on these inputs the feedback neuron fires up to increase

the radius of generalization or resets the radius of generalization. Instead of increasing the radius of generalization, the threshold of the hidden neurons can also be altered.

Training of the Generalized CC4 Network

The training of this network is similar to the training of the CC4 network, excepting that its input vectors have integers. For each hidden neuron $j \in (1, s)$, where s is the number of training samples used to train the network, the weight associated to the links between the input neuron $i \in (1, n)$, where n is the number of input neurons for representing any input vector in the data set, and the hidden neuron j is given by

$$w_{ij} = x_{ij}$$

The weights to the links between the output neurons and the hidden neurons are assigned in a manner similar to the CC4 training algorithm according to the following equation.

$$u_{jk} = \begin{cases} 1 & if \quad y_{jk} = 1 \\ -1 & if \quad y_{jk} = 0 \end{cases}$$

The hidden neurons are all updated with the user specified radius of generalization. This value is set to 0, if there is no user specified value.

Output from the Generalized CC4 Network

After the above network is trained, the new input vectors are presented to the network for generating corresponding outputs. The hidden neuron receives the difference between the input values of the neurons and the associated weights between the input neuron and the hidden neuron. The hidden neuron fires only if the sum of all the values it receives is less than or equal to the user specified radius of generalization. Hence the transition function

of the hidden neuron is the step function as in CC4 network. For each hidden neuron *j* in the network, the sum of the differences of the inputs and corresponding weights is given by

$$f(x) = \sum_{i=1}^{n} \left| x[i] - w_{ij} \right|$$

where, *x* is the new input vector and *n* is the size of input vector. Therefore, the transition function of the hidden neuron is given by

$$g(x) = \begin{cases} 1 & if \quad (r - f(x)) \geq 0 \\ 0 & if \quad (r - f(x)) < 0 \end{cases}$$

Depending on the hidden neurons fired, the output for the given input is generated by the output layer. The activation function or transition function of the neurons in the output layer is also a step function as in the CC4 network. However, if none of the hidden neurons fire, the feedback neuron in the output layer fires as it receives no input. As a result, either the radius of generalization is increased or the threshold is altered and the process repeats until at least a minimum number hidden neuron fires up.

Example 3.1: This example demonstrates the training of the network for the XOR (exclusive-OR) function. This example shows the prescriptive learning capability of the new network, which is similar but simpler than the CC4 algorithm. The truth table for the XOR function is given in table 3.1.

In this example, we do not consider the concept of dynamic radius of generalization. Hence, there is no feedback neuron required in the output layer.

Input X_1	Input X_2	Output Y
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.1: XOR function truth table

Here, we have two bits for input, one bit for output and four training samples. Therefore, the resulting network has two input neurons, one output neuron and four hidden neurons. As, we are training the network with all possible inputs and outputs, there is no need for the radius of generalization. Hence, *r* is set to 0.

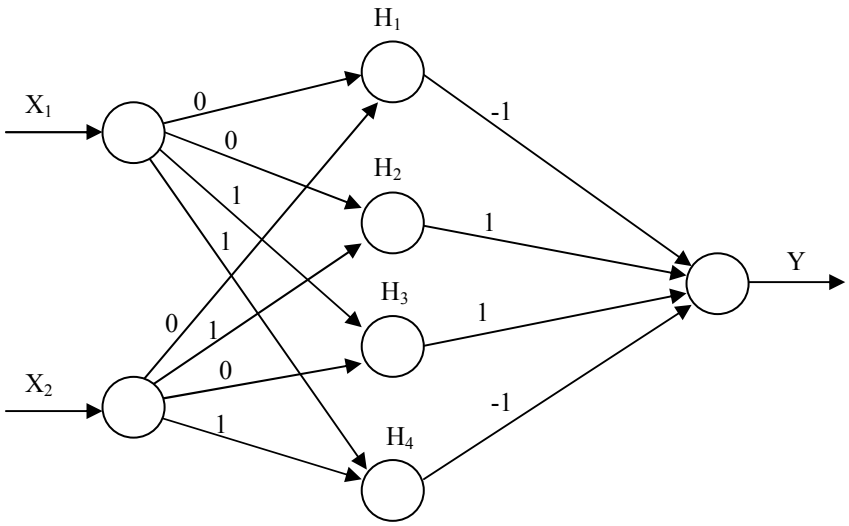


Figure 3.2: Generalized CC4 Network trained with XOR function without feedback

The input weights are simply assigned to the links between the input neuron and the neuron corresponding to the training sample as shown in figure 3.2. The weights to the links between hidden neuron corresponding to the training sample and the output neuron are set in a manner similar to the CC4 algorithm i.e. -1 if the actual output is 0 and 1 if the output is 1. Figure 3.2 shows a completely trained network for the XOR function.

The operation of the network for new inputs after training is given in the table 3.2. Each row represents a new input and the corresponding output. For the x_1 and x_2 values presented to the network, the value of $f(x)$ is given for each hidden neuron H_1, H_2, H_3 and H_4. If $f(x)$ is less than or equal to the radius of generalization r, which is 0 in this case, the output of the corresponding hidden neuron is 1, otherwise the output of is 0. If the dot product of these output vectors with the weights assigned to the links between the output neuron and the hidden layer is positive, the output y is 1, else it is 0.

Inputs		**Inputs to Hidden neurons**				**Hidden neuron outputs**				**Output**
x_1	x_2	H_1	H_2	H_3	H_4	H_1	H_2	H_3	H_4	y
0	0	0	1	1	2	1	0	0	0	0
0	1	1	0	2	1	0	1	0	0	1
1	0	1	2	0	1	0	0	1	0	1
1	1	2	1	1	0	0	0	0	1	0

Table 3.2: Operation of the network for XOR function

An Efficient Implementation of the Generalized CC4 Network

On careful analysis of the CC4 network and the above network with some sample inputs and outputs, it is observed that this algorithm works through all the neurons, for every new input, though only a few neurons within the radius of generalization take part in deciding the output. This results in massive amount of unnecessary operations which cost a lot of CPU time and other resources. To eliminate these unnecessary operations, we present a different approach using a single neuron that mimics the behavior of the new network but uses fewer resources and works faster.

The new approach targets the output of only those neurons that involve in decision making for the new given input from the training. There is no network of neurons required in this approach. The training samples are directly mapped to a neuron or a set of neurons, depending on the implementation, and the output is directly obtained. This approach, however, preserves and is based on the idea of radius of generalization.

This approach can be realized with a single neuron for serial implementation or a set of neurons for parallel implementation. Also, since the complexity of this approach is linear, there is a huge performance gain added to the corner classification approach. Further, this algorithm takes integers as inputs, in contrast to the unary coding used by CC4, which results in very less number of inputs when compared to the inputs to the CC4 algorithm. This algorithm also does not use any kind of multiplication which is computation intensive. It only counts the number of output values of each kind within the radius of generalization which is basic addition and is less CPU intensive, hence, reducing computation cost. As each output is calculated independently, different techniques can be applied to address of the situation where training samples are absent within the user-specified radius of generalization. Hence, the radius of generalization is also dynamic.

The output vector of the training samples can have 1s, 0s or -1s. 0 represents the case where that particular point is not trained, 1 represents a positive result and -1 represents a negative result (0 in the actual training vector). Each output neuron is mapped to a set of training samples based on the radius of generalization. This algorithm works by counting the number of 1s and -1s within the radius of generalization of the neuron for which the

output is required. If number of 1s is greater, the output is 1, otherwise the output is 0. This is nothing but a neuron that takes *m* inputs and gives 1 output with step function as the activation function and 0 as the threshold value. If the count of both 1s and -1s is 0, it means that the there is no training sample within the radius of generalization. In such a case, we dynamically increase the radius of generalization by some desired value, based on the problem specification, to get the output based on the trained sample in contrast to the CC4 algorithm where the output is always 0 in such a case.

Hence, the proposed algorithm uses the basic binary neurons, which take multiple binary inputs and give a single binary output, which makes it a good alternative to the CC4 algorithm, with higher speed of operation and flexibility. The output is based on the dominating neighbors in the training samples within the radius of generalization.

Application to Image Processing

In this section, we show how this approach can be applied to the processing of images. In figure 3.3 the first grid labeled "Sample Image" is the training image. We generate a new image which is labeled as the "Output Image" in the same figure. For easy understanding, only one generated pixel is shown in the output image. For each pixel in the output image, we have a neuron associated that generates the corresponding pixel in the output image. The inputs of each neuron are mapped to all the pixels that are within the radius of generalization of the pixel to be generated. The output is mapped to the pixel to be generated in the output image. All grey pixels correspond to 1, bright pixels correspond to -1 and the black pixels correspond to 0. If the number of grey pixels is greater than the number of bright pixel, the target pixel in the output image is a grey pixel, otherwise the

resulting pixel is a bright pixel. Each pixel is generated similarly using just one neuron or a set of neurons based on the resources available. This ensures high degree of parallelism in the approach.

For color images, a similar architecture and procedure are employed excepting that each pixel is represented in binary and hence we need *n* such mappings, where *n* is the size of each pixel in bits. Due to this approach the output pixel might have one of the parameters of the training pixels in the radius of generalization or could be new pixel generated.

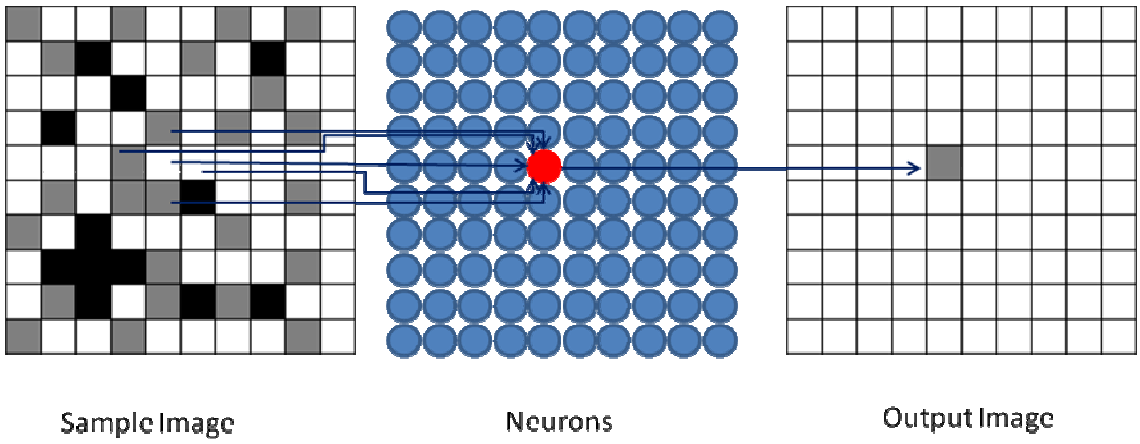


Figure 3.3: Neuron mapping for images

For example, in figure 3.3, the radius of generalization is considered as 1 and the pixel at location (5, 5) is being generated. Since the number of grey pixels within the radius of generalization of this pixel is 3 and the number of bright pixels within the same radius is 2, the resulting pixel is dark.

In general, however, if there are no sample pixels or training pixels present within the radius of generalization, the radius is increased by 1 till a stage is reached where a training pixel is found. For example, consider the pixel at location (8, 3) and radius of generalization 1. It is clear that there is no training sample within the radius of generalization specified. Hence the neuron would gather inputs from the neighboring pixels which are within a radius of 2 and the output would be a bright pixel as there are 6 bright pixels and only 2 gray pixels

Application to Time-Series prediction

Time-series prediction is another area where CC4 algorithm generates impressive results almost instantly. The training samples are generated from the existing data using a sliding window. If *ws* is the window size, then for the first sample, first *ws* number of values are taken as inputs and the next value i.e. *ws*+1 is taken as output. Figure 3.2 illustrates the extraction of training samples from existing data.

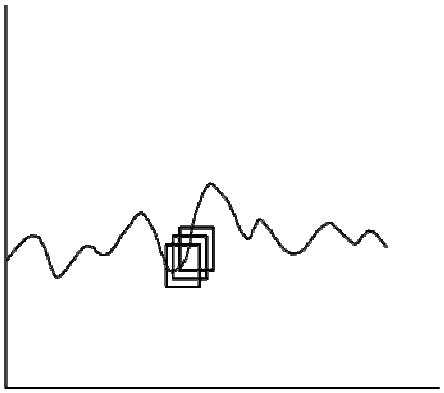


Figure 3.4: Sliding window for extracting training samples

A set of *ts* number of such training samples is called a training set, where *ts* is the number of training samples involved at any time. This set also propagates through the actual data, as it is available, in a sliding window fashion.

At a given time, every sample in the training set is connected to a neuron with an associated weight w_i which is the difference between the radius of generalization and the sum of the absolute values of the differences between the corresponding elements of the input vector. If the weight is negative the neuron does not fire and the output vector of that particular training sample does not participate in the calculation of the next value in the time series.

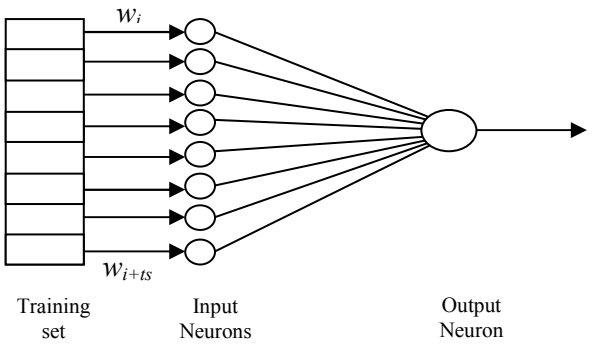


Figure 3.5(a): Basic unit for time-series prediction

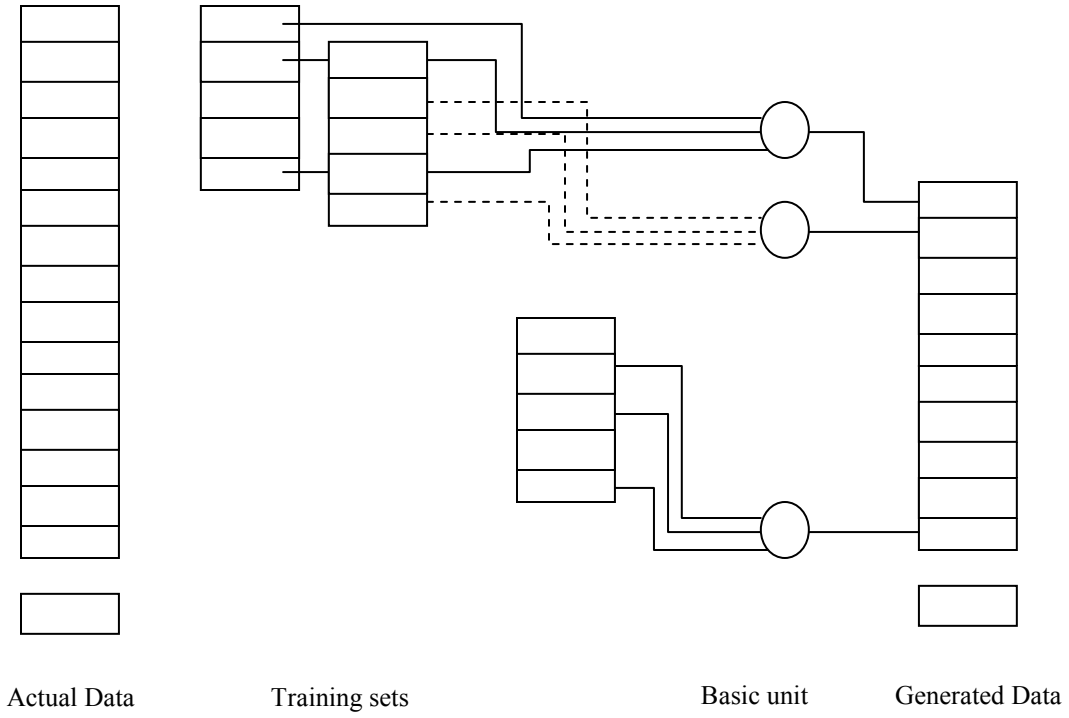


Figure 3.5(b): Architecture for time-series prediction

Figure 3.5(a) illustrates the identification of the training samples used in the process of generation of outputs and figure 3.5(b) illustrates the operation of this approach for time series prediction.

Results

Various experiments were performed with this approach applied on image patterns created with text, black and white bitmap images, and colored bitmap images, which gave expected results. The results of the experiments are presented below.

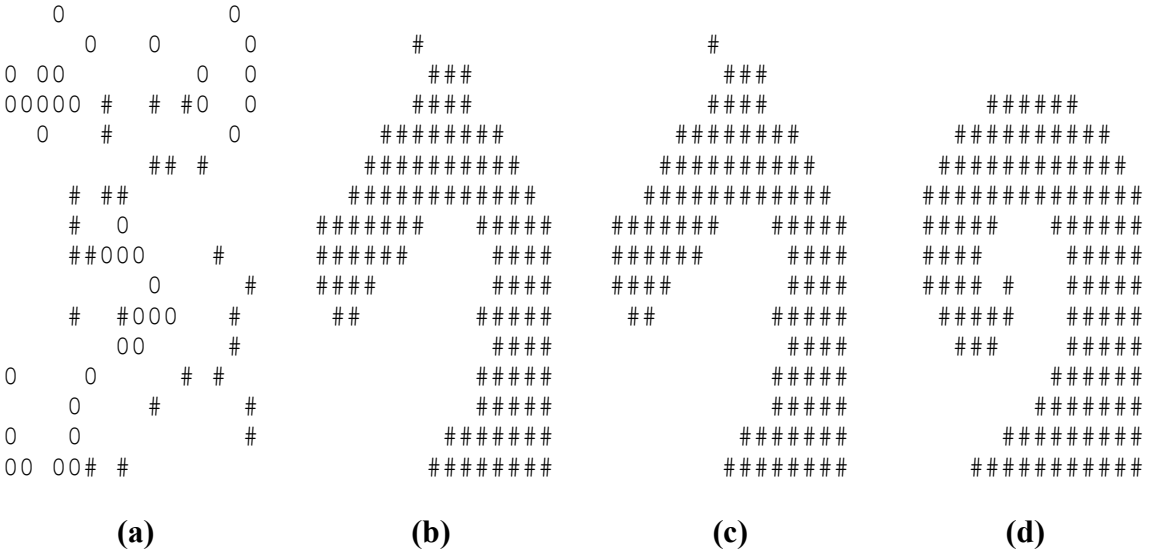
Below is an example of a spiral image pattern created with text. The image pattern is a 16x16 text pattern used for testing the CC4 network [1]. Figure 3.6(a) has all the training samples. 0s represent that the location is learnt negative, # represents the location is learnt positive and all blank spaces represent the locations are not learnt.

The training sample used in this experiment is represented in the form of a matrix as given below.

```
 0  0  0 -1  0  0  0  0  0  0  0  0  0  0 -1  0
 0  0  0  0  0 -1  0  0  0 -1  0  0  0  0  0 -1
-1  0 -1 -1  0  0  0  0  0  0  0  0 -1  0  0 -1
-1 -1 -1 -1 -1  0  1  0  0  1  0  1 -1  0  0 -1
 0  0 -1  0  0  0  1  0  0  0  0  0  0  0 -1  0
 0  0  0  0  0  0  0  0  0  1  1  0  1  0  0  0
 0  0  0  0  1  0  1  1  0  0  0  0  0  0  0  0
 0  0  0  0  1  0  0 -1  0  0  0  0  0  0  0  0
 0  0  0  0  1  1 -1 -1 -1  0  0  0  0  1  0  0
 0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  1
 0  0  0  0  1  0  0  1 -1 -1 -1  0  0  0  1  0
 0  0  0  0  0  0  0 -1 -1  0  0  0  0  0  1  0
-1  0  0  0  0 -1  0  0  0  0  0  1  0  1  0  0
 0  0  0  0 -1  0  0  0  0  1  0  0  0  0  0  1
-1  0  0  0 -1  0  0  0  0  0  0  0  0  0  0  1
-1 -1  0 -1 -1  1  0  1  0  0  0  0  0  0  0  0
```

This matrix is used to train the neural network using the CC4 algorithm. The *radius of generalization* is taken as 4. All the locations are represented in 32 bit (16 for rows and 16 bits for columns) unary coding. Value of each location of this image is then generated from the network so obtained from above procedure. Figure 3.6(b) shows the actual output of the neural network based on the CC4 algorithm. Figure 3.6(c) shows the output based on the single neuron implementation. Figure 3.6(d) is the actual image from which samples are taken.

As it can be seen from figures 3.6(b) and 3.6(c), the results of both CC4 algorithm and the proposed algorithm are identical.



(a) **(b)** **(c)** **(d)**

Figure 3.6: Text image pattern results

Another experiment was performed on a monochrome bitmap image and the results are presented below. Figure 3.7(a) shows the original image. The size of the image used is 127x127 pixels. Figure 3.7(b) is the image generated by the CC4 network based on the training samples from the image with radius of generalization as 2. 50% of the pixels in the image were randomly taken as training samples. Figure 3.6(c) is the image generated

by the single neuron implementation on the same image used for training the CC4 neural network.

Here also it can be seen that the results are identical. However, there is a massive performance gain observed with the new approach. The CC4 implementation took several minutes to generate the results, whereas the single neuron implementation took less than a second to generate the same result. Both of these algorithms were implemented on the same machine. The difference in throughput is due to the complex network generated during training of the CC4 network attributed to the use of unary coding for inputs.

(a) **(b)** **(c)**

Figure 3.7: Monochrome image results

Similar experiments were performed on a color image of Lena which is the standard benchmarking image for testing the performance of neural networks in image processing. In this experiment, we considered a colored bitmap image of size 512x512 pixels. Out of the many experiments performed, we present one example to show the ability of the new network and the single neuron implementation of the new network in image processing applications.

Figure 3.8(a): Original image (512x512 pixels)

Each pixel of the image is represented by 3 bytes or 24-bits of data. 50% of the pixels from the total number of pixels in the image were randomly selected to generate the training sample. The sample image so obtained was used to generate the images shown in figure 3.8(b), 3.8(c) and 3.8(d) with the single neuron implementation of the new network. The radius of generalization for figure 3.8(b) and 3.8(c) was taken as 2 and for figure 3.8(d) it was taken as 0. Figures 3.8(c) and 3.8(d) were generated with dynamic radius of generalization.

Figure 3.8(b): Generated image with static radius of generalization starting at 2

Figure 3.8(c): Generated image with dynamic radius of generalization starting at 2

Figure 3.8(b) has noticeable patches where there is insufficient number of training samples found for generating new pixel within the radius of generalization of 2. Figure 3.8(c) is generated with dynamic radius of generalization and hence has no black spots. Figure 3.8(d) is also generated with dynamic radius of generalization but with 0 as the radius of generalization, i.e. no radius of generalization. It is seen that this gives the best results for such images, however is a bit slower when compared to the other two.

Figure 3.8(d): Generated image with dynamic radius of generalization starting at 0

A series of experiments were performed to see the performance of the algorithm for the time-series prediction. Mackey-glass chaotic time series, which is a commonly used time series for benchmarking neural network performance, was used for these experiments. The equation for discrete time representation of this time series is given by:

$$x(k+1) - x(k) = Ax(k-D)/\{1 + x^{C}(k-D)\} - Bx(k)$$

where A, B, C and D are constants [1].

For these experiments, 1000 data points are generated using the above equation. Out of these 1000 data points first 600 data points are used for training the neural network. Every training sample is a window of four data points for input and one point for output that slides one data point at a time. Below are the results of our experiment in the single neuron approach.

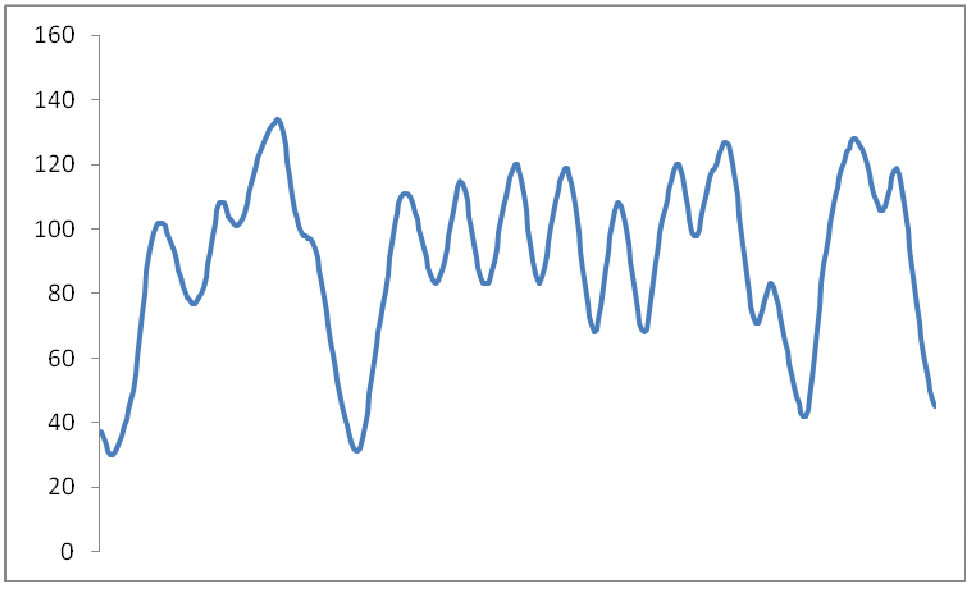


Figure 3.9(a): Actual time series (Mackey Glass Time Series)

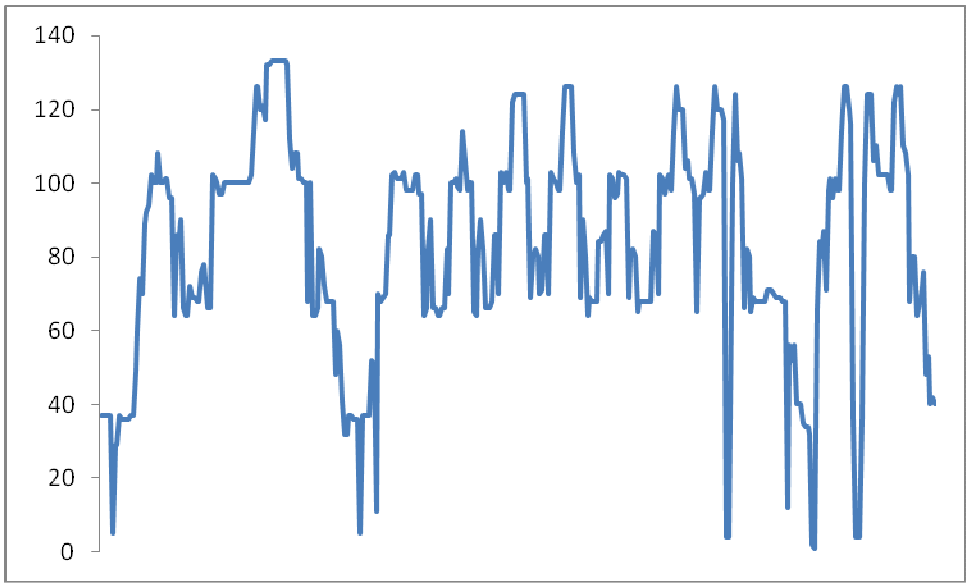


Figure3.9(b): Prediction of actual data value in binary

From these experiments and results obtained it is observed that the new approach is capable of the predicting values of a time series in an effective and efficient way. The best results were observed when the difference in adjacent outputs was considered as output to each sample.

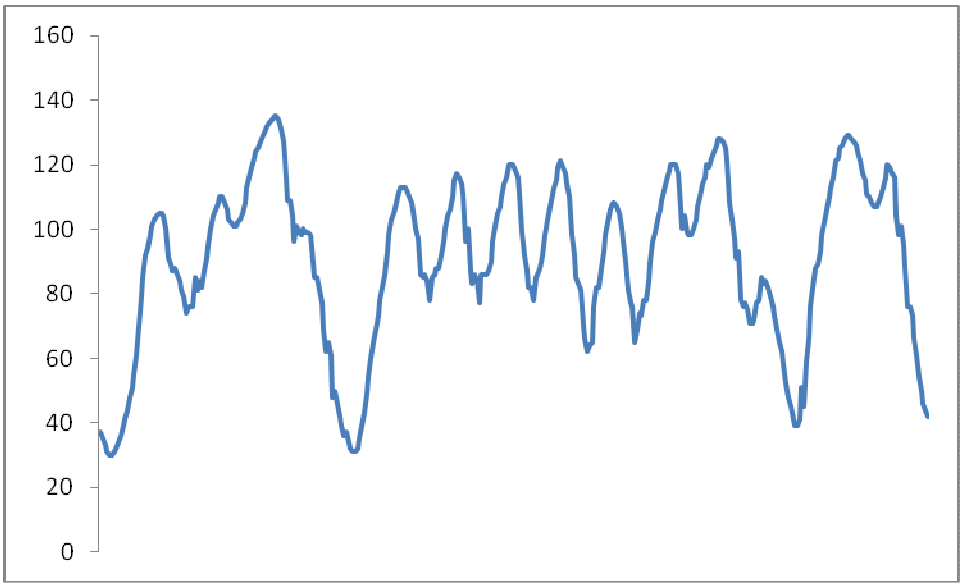


Figure 3.9(c): Prediction based on the difference in adjacent data values

The rms error of the predicted values in 3.9(b) is observed as 20.62. This high value is due to the huge spikes in graph which are an evidence of inadequate training samples within the radius of generalization. In figure 3.9(c), we predict the difference between the previous known value and the new generated value, and add it to the previous known value. This helps in removing the huge spikes and hence is more effective way of predicting time series The rms error in this case is observed to be around 4.62. We present some results in tabular form to show the variation in rms error with different parameters.

Window	Radius	No. of Samples	RMS Error
2	2	50	4.015053
2	6	50	18.41591
2	10	50	17.29573
2	14	50	12.85969
2	18	50	37.78896
2	22	50	41.14106
2	26	50	45.75823

Table 3.3: Time-series prediction with less training and small window size

Window	Radius	No. of Samples	RMS Error
10	2	500	4.427629
10	6	500	4.448114
10	10	500	4.416126
10	14	500	4.349454
10	18	500	4.309709
10	22	500	4.386125
10	26	500	4.498918

Table 3.4: Time-series prediction with heavy training and large window size

Window	Radius	No. of Samples	RMS Error
14	2	200	3.49538
14	6	200	3.49538
14	10	200	3.499529
14	14	200	3.520577
14	18	200	3.504803
14	22	200	3.596576
14	26	200	3.665967

Table 3.5: Time-series prediction with moderate training and large window size

CHAPTER IV

CONCLUSION

We have successfully experimented with the single neuron implementation of the Generalized CC4 neural network and proved that it is a very efficient implementation of the CC4 network which was proposed by Subhash Kak and subsequently granted US patent in 1992[1]. The single neuron approach uses little resources and works in linear time which makes it suitable for use with huge amount of data as opposed to the CC4 algorithm which becomes intractable when used on large problems. Furthermore, the rigidity in the generalization in the CC4 network has been addressed with the new Generalized CC4 neural network by dynamically changing the radius of generalization.

It is observed that the new approach addresses the majority of limitations of the CC4 neural network training while improving the speed of execution and at the same time reducing the amount of resources required, making it ideal for mobile devices and robots where the resources are limited in terms of memory and processing power.

REFERENCES

1. Kun-Won Tang and Subhash C Kak, "A new corner classification approach to neural network training", *Circuits Systems Signal Processing*, Vol.17, No.4, 1998, Pp. 459-469
2. S. Kak, "On generalization by neural networks", *Information Sciences*, vol. 111, pp. 293-302, 1998
3. S. Kak, "A class of instantaneously trained neural networks", *Information Sciences*, vol. 148, pp. 97-102, 2002.
4. P. Raina, "Comparison of learning and generalization capabilities of the Kak and the back propagation algorithms", *Information Sciences*, vol. 81, pp. 261-274, 1994.
5. Kun-Won Tang and Subhash C Kak, "Fast Classification Networks for Signal Processing", *Circuits Systems Signal Processing*, Vol.17, No.4, 1998, Pp. 459-469
6. S. Kak, "Faster web search and prediction using instantaneously trained neural networks," *IEEE Intelligent Systems*, vol. 14, pp. 79-82, November/December 1999.
7. S. Kak, "On training feedforward neural networks", *Pramana J. Physics*, vol. 40, pp. 35-42, 1993.
8. S. Kak, "New algorithms for training feedforward neural networks", *Pattern Recognition Letters*, vol. 15, pp. 295-298, 1994.

9. S. Kak, “Artificial and biological intelligence”, *ACM Ubiquity,* vol. 6, No. 42, pp. 1-20, 2005. Also arXiv: cs.AI/0601052
10. Z. Jihan, P. Sutton, “An FPGA implementation of Kak’s instantaneously-trained, fast-classification neural networks” Proceedings of the 2003 IEEE International Conference on Field-Programmable Technology (FPT), 2003.
11. J. Zhu and G. Milne, “Implementing Kak neural networks on a reconfigurable computing platform," In FPL 2000, LNCS 1896, R.W. Hartenstein and H. Gruenbacher (eds.), Springer-Verlag, 2000, p. 260-269.
12. J. Shortt , J. G. Keating, L. Moulinier, C. N. Pannell , “Optical implementation of the Kak neural network” Information Sciences 171, 2005, p.273-287.
13. R Beale and T Jackson, *Neural Computing, An Introduction*
14. McCulloch, W. S. and Pitts, W. 1988. “A logical calculus of the ideas immanent in nervous activity” In *Neurocomputing: Foundations of Research*, J. A. Anderson and E. Rosenfeld, Eds. MIT Press, Cambridge, MA, 15-27.

VITA

Gangasani Sumanth Kumar Reddy

Candidate for the Degree of

Master of Science

Thesis: GENERALIZATION AND EFFICIENT IMPLEMENTATION OF CC4 NEURAL NETWORK

Major Field: Computer Science

Biographical:

Personal Data:
Born on 14th September, 1983
Education:
Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in December, 2008.

Received Bachelor of Technology degree in Computer Science and Engineering from Jawaharlal Nehru Technological University, Hyderabad, India in May, 2007.

Experience:

Research Assistant to Dr. Subhash Kak, OSU Stillwater, OK.
Dec 2007 – till date

Intern(Web Developer), CREC Stillwater, OK
May 2008 – till date

Intern(Software Engineer), Synopsys Inc. , Hyderabad, India
Jan 2007 – July 2007

Name: Gangasani Sumanth Kumar Reddy Date of Degree: December, 2008

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: GENERALIZATION AND EFFICIENT IMPLEMENTATION OF CC4 NEURAL NETWORK

Pages in Study: 34 Candidate for the Degree of Master of Science

Major Field: Computer Science

The prescriptive learning and generalization capabilities of the instantaneously trained neural networks make them suitable for various applications. Although the CC4 network, which is the most popular of the instantaneously trained neural networks, does well in comparison with the back propagation network, it suffers from several implementation issues when applied to large data sets. To address these issues, we present a generalized CC4 network. We also describe single neuron implementations of the new network which improve the flexibility and performance of the network. We present results of applications of this network to image processing and time-series prediction.

ADVISER'S APPROVAL: _____